# Developer Test - <<SUNG-IL KIM>>

## Goal

Build a Restful Service written in Java or Groovy with the following endpoints:

1. GET : http://<server\_url>/math/add?n1=<numeric param 1>&n2=<numeric param 2>  
     
   This should add numbers 1 and 2 and provide the result in JSON

* Check this Url https://github.com/abstevekim/wetJet\_test/blob/master/add\_get.java

1. POST : http://<server\_url>/math/add (allow for form params 1&2 in a POST body)  
     
   This should add two numbers from a POST body

* Check this Url

https://github.com/abstevekim/wetJet\_test/blob/master/add\_post.java

1. GET : http://<server\_url>/time/now

This should fetch time for MST at time of call from another service (<https://www.developer.aero/WaitTime-API/Try-it-Now> for YYC is a free one) and simplify the result to timezone and current time. Return the result or the timestring in a rational JSON document.

* Check this source code.

@ResponseBody

@RequestMapping("/now")

**public** SessionInfo now(HttpServletRequest req) {

SessionInfo sessionInfo = **new** SessionInfo();

**try** {

sessionInfo.setSessionInfoWithOK(**null**, "result", *callURL*("https://www.developer.aero/WaitTime-API/Try-it-Now"));

} **catch**(Exception ex) {

logger.fatal(ex.getMessage());

sessionInfo.setSessionInfoWithMethodFailure(**null**);

}

**return** sessionInfo;

}

**public** **static** String callURL(String yycTimeUrl) {

StringBuilder yycTime = **new** StringBuilder();

URLConnection urlConn = **null**;

InputStreamReader in = **null**;

**try** {

URL url = **new** URL(yycTimeUrl);

urlConn = url.openConnection();

**if** (urlConn != **null**)

urlConn.setReadTimeout(60 \* 1000);

**if** (urlConn != **null** && urlConn.getInputStream() != **null**) {

in = **new** InputStreamReader(urlConn.getInputStream(),

Charset.defaultCharset());

BufferedReader bufferedReader = **new** BufferedReader(in);

**if** (bufferedReader != **null**) {

**int** cp;

**while** ((cp = bufferedReader.read()) != -1) {

**if** (bufferedReader.read() == "operationDate") {

yycTime.append((**char**) cp);

}

}

bufferedReader.close();

}

}

in.close();

} **catch** (Exception e) {

**throw** **new** RuntimeException("Exception while calling URL:"+ yycTimeUrl, e);

}

**return** yycTime.toString();

}

}

The service must be built using the following technologies:

1. Spring Boot
2. Gradlew (build)
3. Git (source control)
4. Docker (optional - see below)

Please provide all code and answers in a public github (or equivalent) repo. Include a README.md file in the repo with any necessary documentation.

## Additional Questions & Deliverables

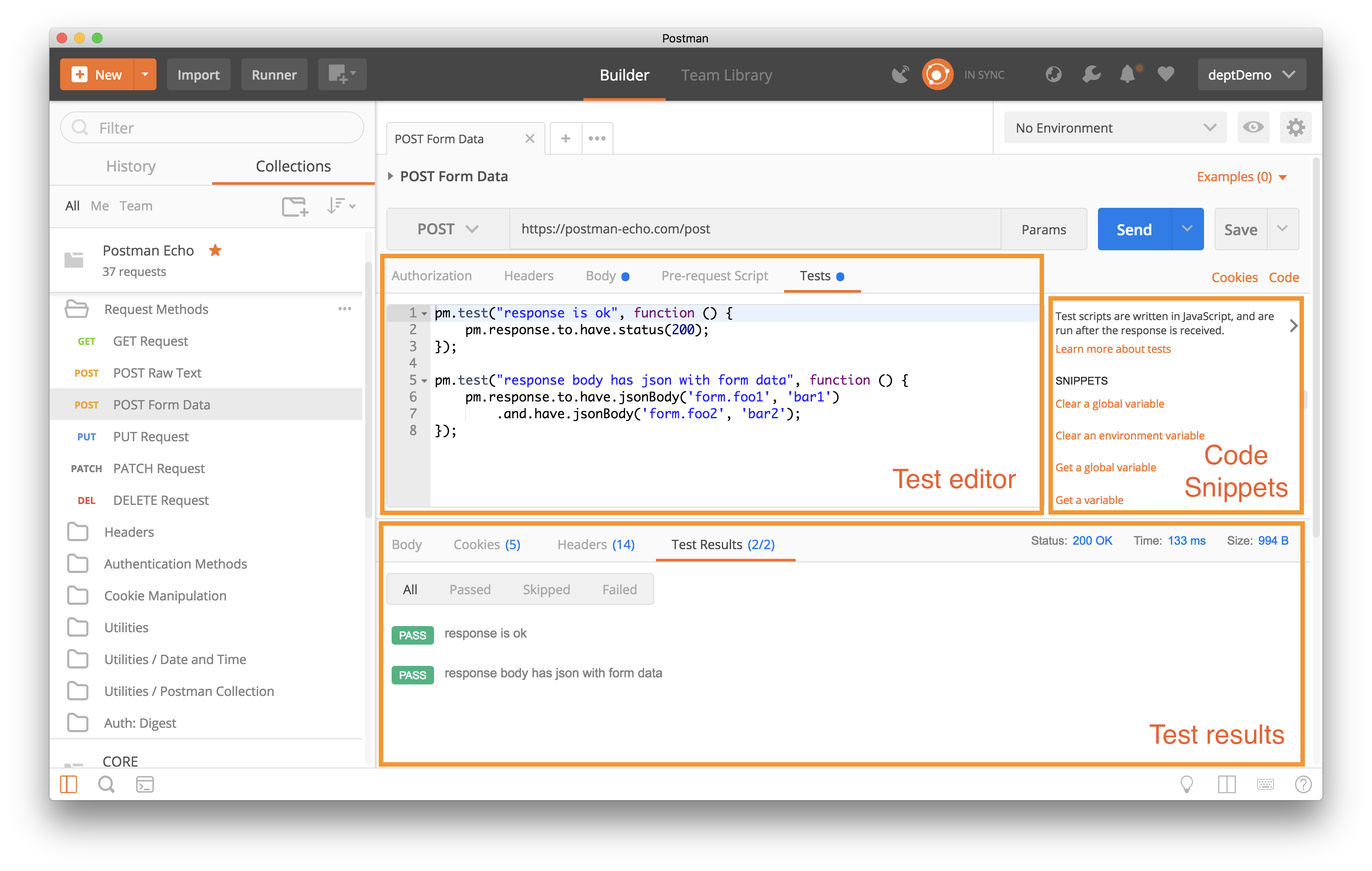
The following is a set of questions and deliverables that should be completed and provided as part of the answer, respectively. If you cannot provide an answer please describe why.

1. All code should be publicly accessible via github url (or similar public git repo).
2. The Rest Service should be deployed to a publicly accessible server on the internet (preferably AWS). Please provide reasons why AWS was not used. If the service is not deployed remotely please provide instructions on how to run it locally. The service should be runnable outside an IDE.

I have no idea.

1. A set of integration tests asserting server functionality ([Postman](https://www.getpostman.com/) test suites, [Rest Assured](http://rest-assured.io/) tests or bash scripts preferred).

* Postman
* With Postman, you can add scripts to your request to use [dynamic variables](https://www.getpostman.com/docs/postman/scripts/postman_sandbox#dynamic-variables), [pass data between requests](https://www.getpostman.com/docs/postman/scripts/postman_sandbox#environment-and-global-variables), and [write tests](https://www.getpostman.com/docs/postman/scripts/test_scripts). Code added under the **Pre-request Script**tab will execute before your request is sent, and code added under the **Tests** tab will execute after your response is received.
* Tests are scripts written in JavaScript that are executed after a response is received. Tests can be run as part of a single request or run with a collection of requests.
* In the Postman app, the request builder at the top contains the **Tests** tab where you write your tests. The response viewer at the bottom contains a corresponding **Test Results** tab where you can view the results of your tests.
* To start building test cases quickly, commonly-used snippets are listed next to the test editor. Select a snippet to append the code to the test editor. If needed, update the stub with assertions specific to your endpoint’s expected response. Then, send the request to view the test results at the bottom.



* Rest Assured

Testing and validating REST services in Java is harder than in dynamic languages such as Ruby and Groovy. REST Assured brings the simplicity of using these languages into the Java domain. For example if your HTTP server returns the following [JSON](http://www.json.org/) at “http://localhost:8080/lotto/{id}”:

{ "lotto":{ "lottoId":5, "winning-numbers":[2,45,34,23,7,5,3], "winners":[ { "winnerId":23, "numbers":[2,45,34,23,3,5] }, { "winnerId":54, "numbers":[52,3,12,11,18,22] } ] } }

You can easily use REST Assured to validate interesting things from the response:

@Test public void lotto\_resource\_returns\_200\_with\_expected\_id\_and\_winners() { when(). get("/lotto/{id}", 5). then(). statusCode(200). body("lotto.lottoId", equalTo(5), "lotto.winners.winnerId", containsOnly(23, 54)); }

.

1. Documentation detailing any service design, security choices regarding api keys or credentials or implementation preferences.

I have no idea.

1. A dockerfile providing packaging as a docker container. If omitted please provide reasoning why Docker was not used.

I have no idea.

1. Explain how caching could be performed on the Restful service and pros/cons

I have no idea.

1. All documentation and code should be provided within the README.md file in github